Title: Exploring Advanced Architectures using Performance Prediction

Author(s): Darren J. Kerbyson
Harvey J. Wasserman
Adolfy Hoisie

Submitted to: Innovative Architecture for Future Generation High-Performance Processors and Systems, IWIA'02, IEEE Computer Society Press

Exploring Advanced Architectures using Performance Prediction

Darren J. Kerbyson, Harvey J. Wasserman, Adolfy Hoisie
Parallel Architectures and Performance Team, CCS-3,
Los Alamos National Laboratory, NM 87545
djk@lanl.gov

Abstract

In this work we show how by the examination of the key characteristics of an application, analytical performance models can be formed. These models are parameterized in terms of computational and communication performances of an individual system and can be used to explore achievable performance of an application prior to system availability. Two applications are considered: an adaptive mesh refinement code on structured meshes, and an Sn transport code on unstructured meshes. These are representative of part of the ASCI workload. One of the models is utilized to validate the performance of a Compaq Alpha-server ES45 supercomputing system being built at Los Alamos, and expected to grow to 30Tera-flops peak performance in the next year. In addition, the models are used to explore the achievable performance on hypothesized future systems with increased peak computation and communication performance.

1. Introduction

The design and implementation of high-performance systems is a highly complex problem requiring knowledge of many factors. The peak performance of a system results from the underlying hardware architecture including processor design, memory hierarchy, inter-processor, communication system, and their interaction. Moreover, the achievable performance is dependent upon the workload that the system is to be used for, and specifically how this workload utilizes the resources within the system. Thus optimizing the peak performance of a system component is only valuable if it has an associated impact on the achievable performance of the workload. Performance analysis is required in order to ascertain the impact on performance resulting from architectural evolution and innovation.

Performance modeling is a key approach that can provide information on the expected performance of a workload given a certain architecture configuration. It is useful throughout a system life-cycle: starting at design when no system is available for measurement, in procurement for the comparison of systems, through to implementation and installation, and to examine the effects of updating a system over time. At each point the performance model should provide an expectation of the achievable performance with a reasonable fidelity. When considering tera-scale systems, a large investment is required throughout the life-cycle and thus performance modeling should be a requirement.

At Los Alamos, there are several performance modeling activities underway that range from detailed simulations that require compute intensive resources to evaluate models [1], through to analytically based performance prediction which can be evaluated rapidly [4,6,7]. In this work we will consider analytically based approaches due to the need to explore different architectural scenarios with reasonable accuracy and time constraints. These approaches do not require a lengthy simulation design or evaluation process.

The approach that we take is application centric. This involves the understanding of the processing flow in the application, the key data structures, and how they use and are mapped to the available resources. From this a performance model is constructed that encapsulates its key performance characteristics. The aim of the model is to provide insight. By keeping the model as general as possible whilst not sacrificing accuracy, it may be used to explore the possible achievable performance in new situations – both in terms of hardware systems and in terms of code modifications. This approach has been successfully used on an adaptive mesh code [6] and a structured mesh transport code [4]. In this work we show how by the examination of the key characteristics of an application analytical performance models can be formed. Two applications are considered, the first is SAGE – an adaptive mesh refinement code which is representative of some of the ASCI workload. The second is Tycho which performs an Sn transport calculation on unstructured meshes. Both applications are described in Section 3.

SAGE has been used as an important component during the installation of a Compaq Alphaserver system at Los Alamos which is expected to grow to 30Tera-flops peak performance. It will be shown in Section 5 that a model can provide an expectation of performance to which actual measurements should be compared. From the experience gained during installation there are many factors in such large scale systems that may not initially function ideally and need to be debugged and rectified. Performance models help to identify when the observed performance does not match the expected achievable performance.

We also show how a performance model can add insight into the possible achievable performance on hypothesized future systems. By considering possible performance improvements in the communication network and computational capabilities of the system, we examine the possible performance improvements that can be achieved by the applications prior to system availability. Thus is one of the main benefits of developing performance models – to be able to explore performance scenarios and to add insight into the achievable performance.

2. Performance Modeling

Goals of performance studies vary, typically depending on the stage of implementation of the application codes and hardware systems. For instance from a software perspective in the early application development stages it may be appropriate to compare alternative design strategies and to recognize their impact on performance in advance of implementation. From a system perspective, it is important to understand the effect that architectural decisions will have on the achievable performance of the workload that the system will to be used for.

A performance model is required in order to explore the possible achievable performance that will result in application or system evolution and innovation. There are two main components that need to be considered in order to obtain a model of performance:

System Characteristics – This includes computational aspects (processor clocks, functional units etc.), the memory hierarchy (cache configuration, memory bus speeds etc.), node configuration (processors per node, shared resources etc.), inter-processor communication (latency, bandwidth, topology), and I/O capabilities.

Workload Characteristics – This includes processing flow, the data structures, their use and mapping to the system resources, their frequency or use, and their potential for resource contention etc.

For modularity and model re-use, the characteristics of the system should ideally be described, and values obtained, independently of any application. Similarly the description of the characteristics of any workload should be described independently of specific systems. Thus, once a model for particular system has been developed, it can be used to examine its performance on a multitude of applications. Similarly, the performance of an application can be compared across systems without any alteration to the application model. This modular approach of hardware and software model separation has been taken in a number of modeling activities include the PACE system [8] for high performance computing, and also INDY [10] for e-commerce based applications.

Hardware and software performance characteristics can be described using a scripting language, which are becoming known generically as Performance Specification Languages (PSL). These can be compiled down into “executable” models which can remain parameterized in a similar way to the actual application. However, current description languages have limitations. They must be flexible in order to describe novel architectural factors which are not present in current systems, be able to describe the characteristics of the workload, and must be able to utilize individual models of system resources which in themselves can quite complex (e.g. [3]).

The workload can be described in a number of ways ranging from statistical behavior of a set of applications through to a detailed understanding of individual applications. The latter is more of a concern to us given that many of the ASCI applications can use 1000’s of processors for a long duration.

The approach we take is thus application centric. It involves the understanding the processing flow in the application, the key data structures, how they use and are mapped to the available resources, and also its scaling behavior. A performance model of the application is constructed from this understanding. The aim is to keep the model of the application as general as possible but parameterized in terms of its key characteristics.

There are several ways in which a model can be constructed and evaluated. One such approach is to obtain a trace of the computation/communication pattern that occurs at run-time. By effectively replaying the trace and costing the time for individual events using hardware specific models, an extrapolation can be made to a new system. This is a detailed evaluation with a potentially high accuracy but loses generality – it is specific to a problem and processor configuration. Such an approach is taken by Dimemas [2].

Our view is that a model should be able to provide insight into the performance of the application on available as well as future systems with reasonable accuracy. Hardware characteristics should not be part of the application model but rather be contained within a component model and be available for use. For instance a component model for inter-processor communication may be parameterized in terms of the message size. The actual model may take on the form of a simple linear analytical expression in terms of latency and bandwidth, or be more complex. Component models may use measurements made by micro-benchmarks, or specified by other means.

An application performance model needs to be validated against measurements made on one or more systems for one or more of its configurations (or data sets). Once a model has been validated it can be used to explore performance and to provide insight in new performance scenarios.

In the following section we detail the salient characteristics of two applications which lead to the formation of a performance model for each. In Section 4, we detail the resource characteristics of a Compaq Alpha-server ES45 supercomputer system. Both application and resource models are used in combination in order to explore achievable performance in Section 5.

3. Application Performance Characteristics

Two applications are considered here which exhibit different performance characteristics. The first is an Adaptive Mesh Refinement (AMR) application known as SAGE. The second is an unstructured mesh application known as Tycho. Both applications are under development at Los Alamos National Laboratory.

In order to develop a performance model of each application, the key processing and scaling characteristics need to be fully analyzed – these are described separately for each code below.

3.1. SAGE – A Structured Mesh AMR Code

SAGE (SAIC's Adaptive Grid Eulerian hydrocode) is a multidimensional (1D, 2D, and 3D), multimaterial, Eulerian hydrodynamics code with adaptive mesh refinement (AMR). It comes from the Los Alamos National Laboratory Crestone project, whose goal is the investigation of continuous adaptive Eulerian techniques to stockpile stewardship problems. SAGE represents a large class of production ASCI applications at Los Alamos that routinely run on 1,000’s of processors for months at a time.

Adaptive mesh refinement operations are performed on cells as necessary at the end of each cycle in the processing. Each cell at the top most level (level 0) can be considered as root node of an oct-tree of cells in lower levels. For example, the shock-wave indicated in the 3-D spatial domain in Figure 1 by the solid line may cause cells close to it to be split into smaller cells. In this example, a cell at level 0 is not refined, while a cell at level n is a domain 8^n times smaller.

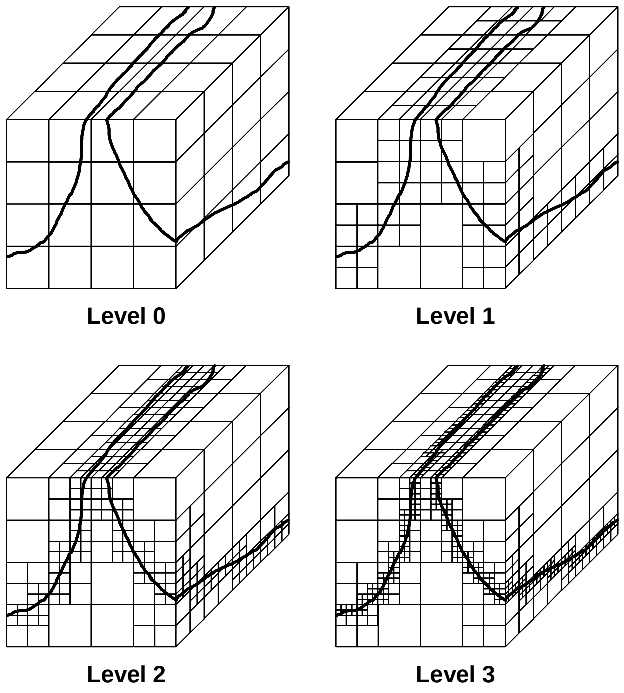

Figure 1. Example of Adaptive Mesh Refinement at multiple levels

3.1.1 Key Performance Characteristics of SAGE. The key characteristics of SAGE are:

Data decomposition – SAGE uses a spatial discretization of the physical domain utilizing Cartesian grids. This spatial domain is partitioned across processors in sub-grids such that the first processor is assigned the first *E* cells in the grid (indexed in dimension order – X,Y,Z), and so on. This results in a 1-D *slab* partitioning across processors. The problem size grows proportionally with the number of processors in the normal operational mode of SAGE, i.e. a weak-scaling characteristic.
Processing flow – the processing proceeds in cycles. In each cycle there are a number of stages that involve the three operations of: one (or more) data gathers to obtain a copy of remote neighbor data, computation on each of the gathered cells, and one (or more) scatter operations to update data on remote processors.
AMR and load-balancing – at the end of each cycle, each cell can either be split into smaller 2x2x2 cells, combined with its neighbors to form a single larger cell, or remain unchanged. A load-balancing operation takes place if any processor contains 10% more cells than the average cells across all processors.

The 1-D slab decomposition leads to two important factors that influence the achievable performance. Firstly, the amount of communication increases as the number of processors increases. This is due to the boundary surface of a sub-domain in a 1-D decomposition scaling at the 2/3 power of the number of processors. Secondly, since the number of cells per processor is constant, there is a point at which the sub-domain will be only a single cell wide, and also when a single slab is mapped to more than one processor. At this point the gather/scatter communications will not be just between adjacent processors but rather between processors a certain distance apart. This distance is actually equal to the number of processors sharing a single slab. This distance will be reflected in the number of simultaneous out-of-node communications that will contend for the communication channel. The maximum number of processes contending will be equal to the number of processors within a node. Full details on the characteristic scaling behavior of SAGE is given in [6].

3.1.2 Performance Model of SAGE. The performance model of SAGE consists of three main components: computation, memory contention within a node, and inter-node communication. The run-time for one cycle of the code can be modeled as:

$$\begin{aligned} T_{cycle_i}(P, E, \mathbf{D}, \mathbf{A}, \mathbf{M}_{\mathbf{cm}}) = \; & T_{comp}(E.D_i) + T_{memcon}(P, E.D_i) + \\ & T_{allreduce}(P) + T_{GScomm}(P, E, D_i) + \\ & T_{divide}(A_i) + T_{combine}(E.D_i) + \\ & T_{load}(M_{cm_i}, P) \end{aligned}$$

where

$T_{comp}(E.D_i)$	- the computation time
$T_{GScomm}(P,E,D_i)$	- the gather/scatter communication time
$T_{allreduce}(P)$	- the allreduce communication time
$T_{memcon}(P,E.D_i)$	- the memory contention that occurs between PEs within a node
$T_{divide}(A_i)$	- the time to divide cells in iteration *i*
$T_{combine}(E.D_i)$	- the time to combine cells in iteration *i*
$T_{load}(M_{cmi}, P)$	- the time to perform the load-balancing

The model consists of an additive sum of communication and computation components since the local gather/scatter communications effectively synchronize the processing across processors.

The gather and scatter communication time is the time taken to provide boundary information by processors owning boundary data. This is related to the 1-D slab decomposition and the number of processors that share a single slab. The size of the boundary data in each direction, $Surface_x$, $Surface_y$, $Surface_Z$, are calculated as a function of the number of cells per processor and the processor count. The frequency of this operation was measured to be 160 floating point and 17 integer gathers and scatters in total per cycle. The contention on the communication channel per node is also modeled – this is taken to be a multiplicative factor on the communication time representing the number of processors performing simultaneous out of node communications. This factor is also a function of the number of cells per processor, and the processor count.

The memory contention represents the extra time required per cycle when multiple processors within a node contend for memory. This can be measured by considering different configurations of processors for the same problem – for instance using all processors with a node, or using 1 processor in each of P_{SMP} nodes (P_{SMP} is the number of processors per node). The difference in execution time is approximately the additional time due to memory contention (assuming the communication time is small).

The AMR operation is modeled from a number of time histories of values defined on a cycle by cycle basis which can be measured for a particular calculation. These time histories represent the level 0 cell division factor (**D**), the maximum number of cells added over all processor by the division process (**A**), and the maximum number of cells moved from a single node (**M_{cm}**) in the load balancing operation.

Table 1. SAGE model input parameters (M – measured, S – specified)

Category	Type	Parameter	Description
Application	S	E	Cells per processor
	M	$\boldsymbol{D, A, M_{cm}}$	AMR Time histories
Mapping	S	$Surface_x$, $Surface_y$, $Surface_Z$	Surface size (in cells) of the sub-grid on each processor (in 3-D)
System	S	P	Number of processors
	S	P_{SMP}	Processors per SMP box
	S	C_L	Communication Links / node
	M	$L_c(S)$, $B_c(S)$	Latencies and Bandwidths achieved in one direction on bi-directional communication
	M	$T_{comp}(E)$	Sequential cycle time of SAGE on *E* cells
	M	$T_{mem}(P)$	Memory contention

The parameters used in this model are listed in Table 1 according to whether they are application, system, or mapping parameters. Further details on the formation of this model can be found in [6].

3.2. Tycho – An Unstructured Mesh Code

Tycho is currently in development at Los Alamos for exploring Sn transport calculations on unstructured meshes [9]. It is estimated that deterministic particle transport on structured meshes accounts for over 50% of the execution time of many realistic simulations on current DOE systems.

The processing in Tycho corresponds to a wavefront technique using an iterative sweeping method. A wavefront of processing propagates across the mesh in all sweep directions contained within a discrete ordinates set. The calculation results in a 'software pipeline' of computation in each of sweep direction [4], and results in several interesting performance characteristics. The investigation into the performance of this calculation on structured meshes has been examined in some detail [9].

Each sweep direction results in a specific cell processing order – determined by the mesh geometry. An example is shown for two sweep directions, Ω, in Figure 2 on a small 2-D unstructured mesh. A total of 5 steps of the processing are shown. The edge of the sweep corresponds to a wavefront and is shown as black. It requires the grey cells to have been processed in previous steps. The same operation can take place in three dimensions resulting in a 'wavefront surface'.

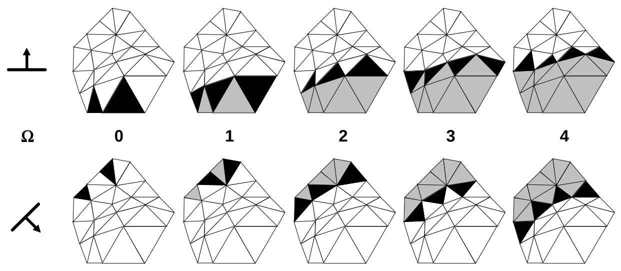

Figure 2. Example sweep processing on a small unstructured mesh

An example 2-D partitioning of an irregular mesh is shown in Figure 3. The communications between processors are shown by arrows, and a simplified propagation of the sweep is shown by the grey lines. The sweep starts in the top left corner and propagates to the lower right. Each grey line indicates the progress of the sweep in each step in this propagation. Tycho actually enables all directions to commence simultaneously.

The sweep processing is also blocked – a maximum number of cell-angle pairs that can be processed per step is specified by the *MCPS* input. This blocking helps to alleviate the starvation in the direction of the sweeps However some starvation will remain on processors in the pipeline due to insufficient boundary data being received to enable the processing of *MCPS* cells.

To minimize processor starvation cells are assigned a priority. Cells with high priority are processed first. The key in this is the priority assignment. In general processor boundary cells will be of high priority (these need to be processed in order that cell fluxes can travel down the pipeline), and cells needing to be processed prior to boundary cells are given an even higher priority. This approach attempts to maximize cell boundary production – different schemes have been analyzed using Tycho [9].

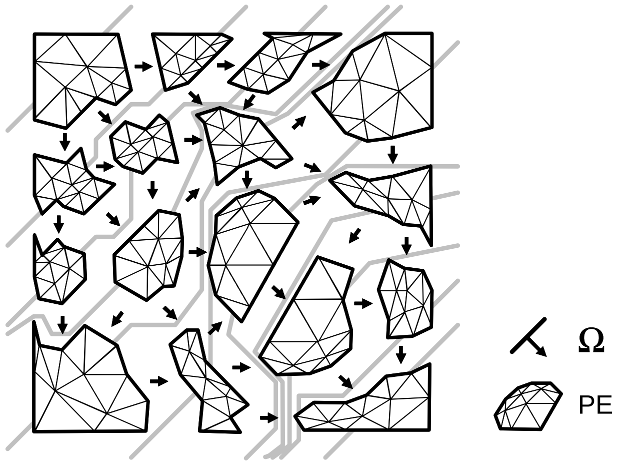

Figure 3. Example partitioning and sweep flow on a 2D unstructured mesh

3.2.1. Key Performance Characteristics of Tycho. The key processing characteristics in Tycho are:

Data Decomposition – the mesh in Tycho is partitioned in 3-D using Metis [5]. This partitioner aims to produce equally sized partitions whilst minimizing boundaries, i.e. keeping the work across processors constant whilst minimizing the communication time. In an idealized mesh, all partitions would have the same number of cells, *Ep*, be hexahedral in shape with minimum surface-to-volume ratio and have six nearest neighbors.

Pipeline Processing - All sweeps in the discrete ordinate set commence at the start of a Tycho iteration. The first elements processed are those that lie on the boundary with no inflows in the sweep direction. Thus sweeps generally start from the surface of the mesh and work their way to the centre before propagating out the opposite side. There are two components in this, namely the propagation of the sweep from one side of the mesh to the other (the so called pipeline length), and the total work that is done on each mesh partition. In an idealized mesh, the pipeline length is $P_L = (P_x-1)+(P_y-1)+(P_z-1)$ where P_x, P_y and P_z are the number of processors in each of the three dimensions respectively. Also the total work done on each mesh partition is equal to: $Wp = Ep*|\Omega|$, where $|\Omega|$ denotes the number of sweep directions.

Process Starvation - Each step in Tycho consists of three stages: process cells at the top of the priority queue, send boundary data to downstream PEs, and receive boundary data from upstream PEs. The maximum amount of work done in a step on each PE is determined by the input parameter *MCPS* (MaxCellsPerStep). The processing situation is made complicated by the dependence between upstream and downstream cells in the sweep directions. There will almost always be a degree of inefficiency in this operation and processors will be starved of work waiting for the results from other PEs. This inefficiency can be quantified by using the metric of Parallel Computational Efficiency (*PCE*) [9]:

$$PCE = \frac{W_p}{\sum_{i=1}^{\#steps} \max_p \left(\left| work(i,p) \right| \right)}$$

The *PCE* represents the fraction of the maximum cells that are processed over all steps in an iteration. Each mesh will have a specific value of *PCE* on each processor configuration. However, in the general case a value of the *PCE* has to be assumed – possibly based on experience from prior meshes. This assumption can be inaccurate reflecting the abstraction that a model needs in order to be generally applicable.

Strong Scaling – Tycho exhibits a strong scaling characteristic, the input mesh size is constant and thus partitions become smaller on larger processor counts. The number of cells mapped to each processor thus changes on different processor counts, and can also effect the performance obtained from the memory hierarchy. For instance when a mesh partition becomes small enough to fit in cache the performance will be better then if main memory is utilized.

3.2.2 A Performance Model of Tycho. In this model we assume that the three stages of a Tycho step are distinct and do not overlap – those of computation, blocking sends, and blocking receives. This is a simplification which allows an analytical model to be formulated but also incorporates some degree of inaccuracy. The time for an iteration of Tycho can be given by:

$$T_{iter}(N,P) = \#Steps.\left(Work_{step}.T_{Elem}(E_P) + 6.T_{BComm}(N,P) \right)$$

where

#Steps	– number of steps in an iteration,
$Work_{step}$	– number of cells processed in a step
$T_{Elem}(x)$	– time to process a cell-angle pair given a total of *x* cells mapped to a processor.
$T_{BComm}(N,P)$	– time to communicate boundary information on a mesh of size *N*, on *P* processors.

The first term in this model represents the computation and the second term communication. The number of steps is assumed to be given by:

$$\#steps = \left(\frac{Ep * |\Omega|}{MCPS * PCE} \right) + (P_x - 1) + (P_y - 1) + (P_z - 1)$$

where E_p is the number of cells per PE (assumed constant at *N/P*), *MCPS* is the MaxCellPerStep Tycho input parameter, $|\Omega|$ is the number of sweep directions, and *PCE* is the Parallel Computational Efficiency as described earlier. P_x, P_y and P_z are the number of processors in the logical x, y, and z dimensions respectively. The first part of this equation represents the number of work steps and the second part the pipeline length.

The work on each PE in each step is assumed a constant, and taken as the minimum of *MCPS* and the total number of cell-angle pairs on a processor. The communications are also assumed constant – six neighbor communications per processor per step, of size $E_P^{2/3}$ (assumes an idealized cube).

These assumptions will generally lead to an under-prediction by the model, but it has been found to be reasonably accurate with an average error of 9% over many mesh-processor configurations. The parameters used in this model are listed in Table 2. Note that most of the system parameters are common with those for SAGE.

Table 2. Input parameters to the Tycho performance model (M : measured, S : specified)

Category	Type	Parameter	Description
Application	S	*N*	Total cells in the mesh
	S	$\|\Omega\|$	Number of sweep directions
	S	MCPS	Max cells processed per step
	S	PCE	Parallel compute Efficiency
System	S	*P*	Number of processors
	S	P_{SMP}	Processors per SMP box
	S	C_L	Communication Links / node
	M	$L_c(S)$, $B_c(S)$	Latencies and Bandwidths achieved in one direction on bi-directional communication
	M	$T_{Elem}(E_P)$	Time to process a cell given E_P cells per processor

4. The Compaq Alpha-Server ES45

The system considered here consists of 512 Compaq Alpha-Server ES45 nodes currently installed at Los Alamos National Laboratory. This system is expected to grow to 30Teraflops peak performance over the next year. Each node contains 4 Alpha Ev68 processors running at 1GHz which are internally connected using two 2GB/s memory buses to 16GB of main memory. Each processor has an 8MB unified level 2 cache, and 64KB L1 data cache. The Alpha processor has a peak performance of 2 floating point operations per cycle. Thus current installation has a peak performance of 4Tflops/s.

Nodes are interconnected using the Quadrics QsNet high-performance network. This network boasts high-performance communication with a typical MPI latency of 5µs and a throughput of up to 340MB/s in one direction (detailed performance figures are discussed in Section 4.1). The Quadrics network contains two components – the Elan network interface card, and the

Elite switch. The Elan/Elite components are used to construct a quaternary fat-tree topology. Example fat-tree networks of a dimension 2 and 3 are shown in Figure 4. A quaternary fat-tree of dimension n is composed of 4^n processing nodes and $n.4^n$-1 switches interconnected as a delta network. Each Elite switch contains an internal 16x8 full crossbar. A detailed description of the Quadrics network can be found in [11].

In order to implement a single rail (a single fat-tree network) a single Elan PCI interface card is used per node in addition to a number of Elite switch boxes. The Elite switches are packaged in 128-way boxes. The first level of boxes implements the first 3 levels of the fat-tree and consists of 64 down and 64 up ports. The second level of boxes implements a further two levels of the fat-tree resulting a possible system of size 1024 nodes. Further levels may be used to implement larger systems.

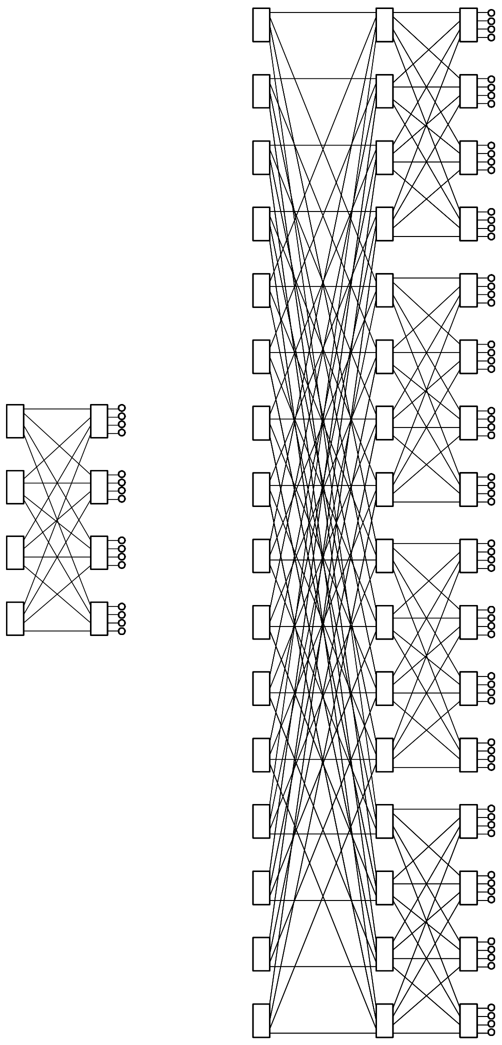

Figure 4. Network topologies for a dimension 2, and 3 quaternary fat-tree

The system being installed at Los Alamos actually contains two rails of the Quadrics network, i.e. two parallel independent networks using two Elan cards per node, and two complete sets of Elite switches.

4.1. System Performance Characteristics

The performance of the system is considered in isolation from any specific workload where ever possible. These characteristics include: computational, memory, intra-node and inter-node communication, and I/O performance. We consider here the characteristics that are relevant to the two applications described in Section 4. These characteristics are listed in both Table 3 and 4.

Table 3 includes general characteristics of the system along with application performance parameters which were measured on a single node. Table 4 includes detail of point-to-point communication performance as observed on a multi-node system using micro-benchmarks. A linear model for the communication time is assumed which uses the latency (L_c) and Bandwidth (B_c) of the communication network for varying sizes of messages (S). These are listed for two configurations of the ES45 internal PCI bus – namely for both a 33MHz and a 66MHz bus. As will be seen in Section 5, both configurations were used as input to the SAGE performance model.

Table 3. General system, and application specific performance parameters for the Compaq ES45.

Category	Parameter	Value
System	P	2048
(General)	P_{SMP}	4
	C_L	2
SAGE	$T_{comp}(E)$ (s)	0.38
	$T_{mem}(P)$ (s)	$\begin{cases} 1.8 & P = 2 \\ 4.8 & P > 2 \end{cases}$
Tycho	$T_{Elem}(E_P)$ (µs)	$\begin{cases} 7.0 & E_P \geq 16K \\ 0.98Ln(E_P)-3.4 & 800 < E_P < 16K \\ 3.0 & E_P \leq 800 \end{cases}$

Table 4. Inter-node communication performance parameters for the Compaq ES45

Parameter	33 MHz PCI bus	66 MHz PCI bus
$L_c(S)$ (µs)	$\begin{cases} 9.00 & S < 64 \\ 9.70 & 64 \leq S \leq 512 \\ 17.4 & S > 512 \end{cases}$	$\begin{cases} 6.10 & S < 64 \\ 6.44 & 64 \leq S \leq 512 \\ 13.8 & S > 512 \end{cases}$
$1/B_c(S)$ (ns)	$\begin{cases} 0.0 & S < 64 \\ 17.8 & 64 \leq S \leq 512 \\ 12.8 & S > 512 \end{cases}$	$\begin{cases} 0.0 & S < 64 \\ 12.2 & 64 \leq S \leq 512 \\ 8.30 & S > 512 \end{cases}$

5. Use of the Performance Models

The performance models for both SAGE and Tycho have been previously validated on large scale systems including several ASCI machines and the CRAY T3E. Typical prediction error for SAGE was found to be 5%, and that for Tycho was 9%. This validation enables the models to be used in new scenarios to explore performance and to provide insight.

In this section we consider two separate performance cases. The first is the use of SAGE to validate the performance that was observed during the installation of the 512 node Compaq System at Los Alamos. The second case explores the performance of both SAGE and Tycho that may be achieved on future large-scale systems with improved computation and communication performance.

5.1. System Installation

The SAGE model as described in Section 3, was used to provide an expectation of the performance of the Compaq Alpha-Sever ES45 system (as described in Section 4) prior to installation. The model required a number of measurements, as listed in Tables 3 and 4, which were obtained on a small 8 node system.

Two performance scenarios were considered prior to the installation of the system. These differed only in the speed of the internal PCI bus of the Compaq ES45 nodes, which were initially set at 33MHz, and later upgraded to 66MHz. The speed of the PCI bus determines the available bandwidth between the Quadrics NIC and the processor memory and thus can have a significant impact on the performance of any parallel application.

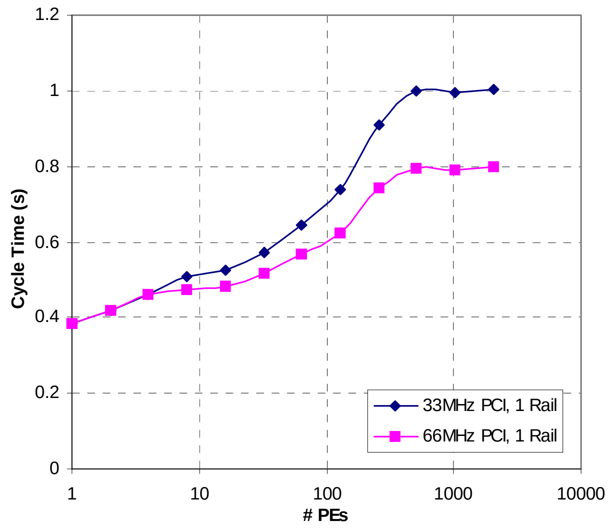

Figure 5. Performance prediction of SAGE on a Compaq ES45 system with QsNet

The expected performance of the system is shown in Figure 5 for both PCI bus speeds. The performance model predicts the cycle time of SAGE. The cycle time would be constant across all processor configurations if an idealized speedup was obtained. This is due to the weak scaling use of SAGE. It can be seen that when using an ES45 with a 66MHz PCI bus in comparison to a 33MHz bus the performance of SAGE is improved by at most 20%. The cycle time is predicted to plateau at above 512 processors – this is the point at which all gather/scatter communications are out-of-node leading to a maximum contention for the NIC.

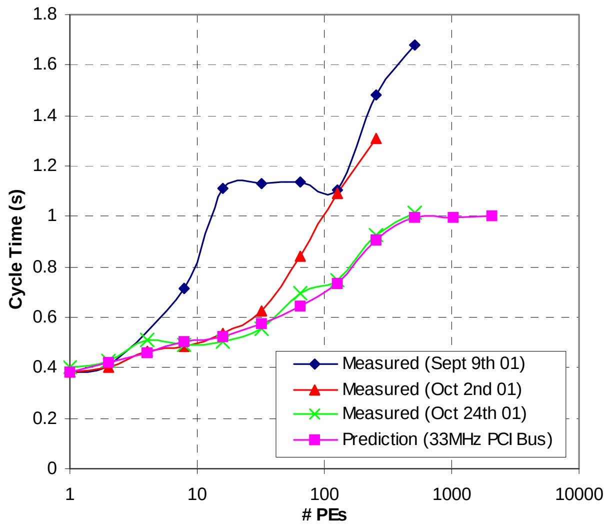

Figure 6. Measured performance of SAGE compared with predictions (33MHz PCI Bus)

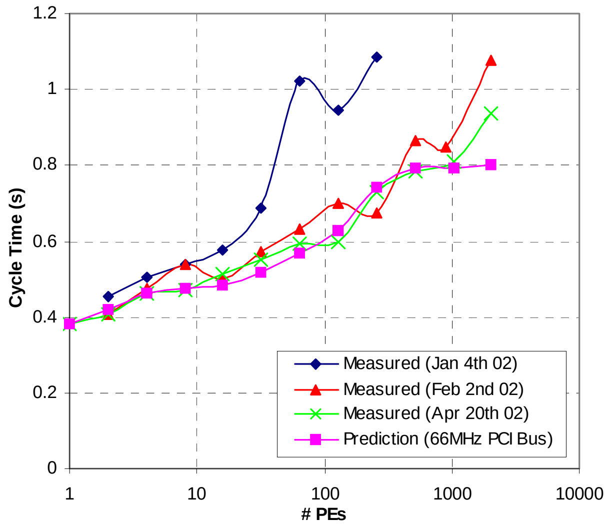

Figure 7. Measured performance of SAGE compared with predictions (66MHz PCI Bus)

The performance of SAGE was measured at several points after the initial installation of the machine had taken place. The obtained performance is shown in Figure 6 on several dates. The difference in performance between the test runs can be attributed to the identification and debugging of faults during the installation process. As one would expect with a large-scale system everything does not work ideally from the first system boot. Several faults were identified including some poorly performing inter-node connections, and also several software issues which lead to an O/S upgrade.

After the installation debugging process it can be seen in Figure 6 that the system achieved the expected performance predicted in advance by the model. A similar process was followed when the nodes were upgraded to a 66MHz PCI Bus. During this process it was found that some nodes reverted back to their original 33Mhz status and thus effected achievable application performance. Several measurements were again taken during this upgrade with the expected performance available from the model. These are compared in Figure 7.

It can be seen from both Figure 6 and 7 that it was only after all the upgrades and system debugging had taken place that the measurements matched the expected performance. Without the model, it would have been difficult to ascertain if the application was achieving a reasonable performance or not. When differences occurred between the model and measurements, further low-level kernel tests were executed on the computational nodes, and the communication network to help identify the source of the problem.

5.2. Future Systems

The performance models for SAGE and Tycho can be used to explore the possible performance that may be observed on future systems. In the following analysis we assume that the system architecture is similar to that of the Compaq Alpha-server ES45 in that there are a number of nodes connected by a Fat-tree topology. The nodes are also assumed to contain four processors. The performance of each application is examined on this hypothesized system when assuming that each of the computational performance, the network bandwidth, and the latency have improved in performance by a factor of eight. This is done be altering the system input parameters to the model as listed in Tables 3 and 4. For instance to improve the latency by a factor of 8, the values for $L_c(s)$ were reduced.

Figure 8 shows the performance of SAGE in terms of the time to process one cycle when the performance of individual system components are changed. In addition, the performance improvement is shown in Figure 9 along with the improvement that could be obtained by simply doubling the number of processors used (with no sub-system performance improvements).

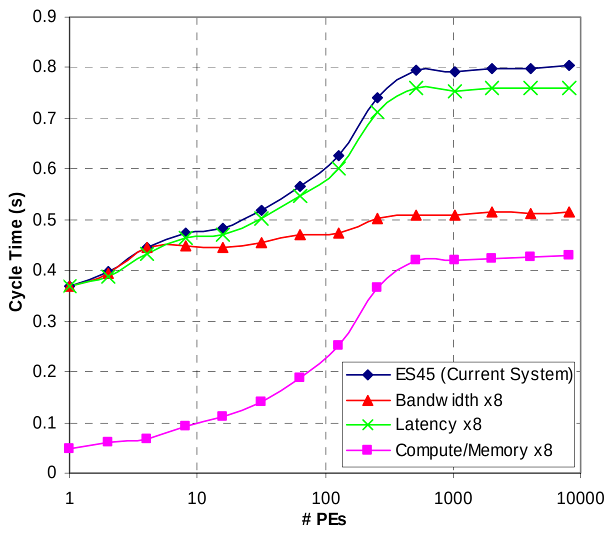

Figure 8. Performance prediction of SAGE when considering individual improvements in sub-system performance.

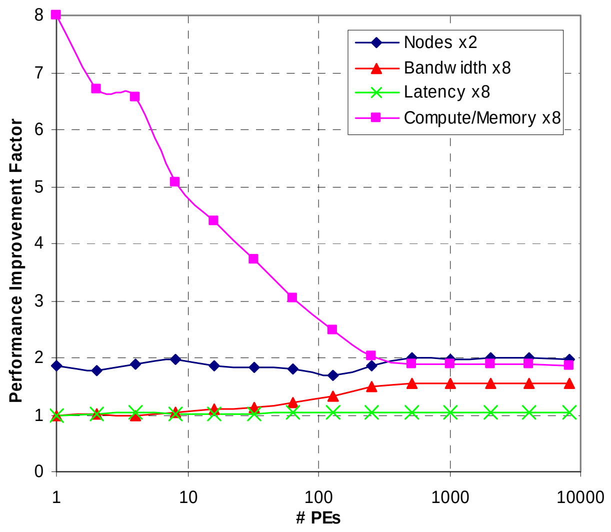

Figure 9. Performance improvement of SAGE with changes in sub-system performance

It can be seen that SAGE is actually computation/memory bound on the lower number of processors. Thus, improving just the performance of a node will give the greatest performance improvement in this region. On larger number of processors the greatest performance improvement could be gained by simply doubling the number of processors. This is due to the plateau in the cycle time occurring on larger processor counts.

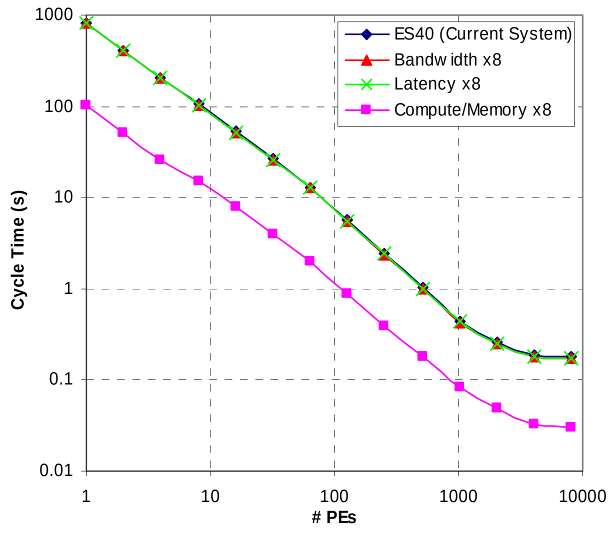

Figure 10. Performance prediction of Tycho when considering individual improvements in sub-system performance.

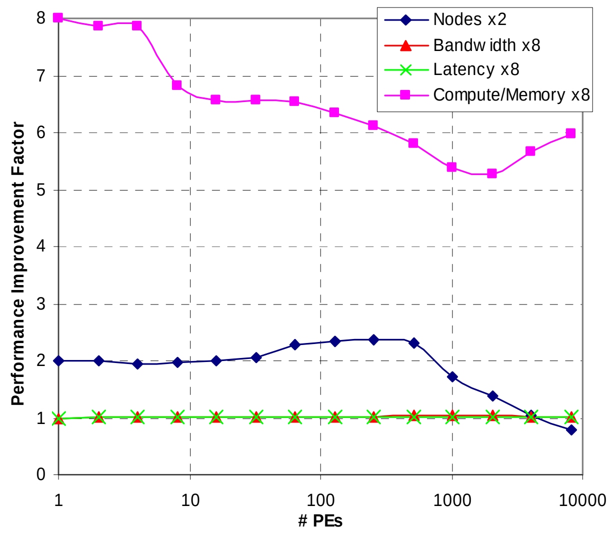

Figure 11. Performance improvement of Tycho with changes in sub-system performance

A similar analysis is depicted in Figures 10 and 11 for Tycho. This is performed for a mesh with 1,000,000 cells, and assumes that a *PCE* of 0.9 is achievable in all cases. The form of the curves in Figure 10 is different to that of SAGE due to its strong scaling characteristic. Tycho remains compute bound throughout. As can be seen in Figure 11, the best performance improvement would be gained by an increase in node computational performance. These curves do not directly state the efficiency of the calculation – on larger processor counts the efficiency actually decreases. It can be seen that using more than 4000 processors is not beneficial as it does not result in any further performance improvement. This is due to the pipeline length dominating the performance on this mesh size. The pipeline length increases as the number of processors increase.

These studies of the expected performance on possible future system illustrate the power of performance modeling. They can be used to explore these performance scenarios giving a good indication of the performance that should be achievable by each application. It should be noted however, that the performance improvements vary from application to application. If all elements of the workload to which a system is to be used for are known in advance, the information from many performance models can be combined to give quantitative information on the expected performance of the full workload. Such information may be useful in system architecture designs.

6. Conclusions

We believe that performance modeling is the key to building performance engineered applications and architectures. Models adds insight into the performance of current systems, reveal bottlenecks and show where tuning efforts would be most effective. They also allow the performance on future systems to be explored. The latter is important for both application and system architecture design as well as for the procurement of supercomputer architectures

In this work we have described how the key characteristics of two applications were combined into performance models.

We have shown that an accurate performance model can be used to validate the performance of a system during its installation. The performance model of one of the ASCI codes, SAGE, has been utilized here to provide an expectation of the runtime on the system prior to its availability. When installing a new system there are often a number of refinements that need to be done in both the software system, and hardware components, before the machine operates at the expected level of performance. The performance model for SAGE has been shown to be of great use in this process. The model has effectively provided the performance and scalability baseline for the system performance on a realistic workload. Initial system testing showed that its performance was almost 50% less than expected. After several system refinements and upgrades over a number of months, the achieved performance matched exactly the expectation provided by the model. Thus performance models can be used to validate system performance.

In addition the performance models have been used to explore the possible achievable performance on future, hypothesized, architectures. This demonstrates an important use for developing models – that of providing a means to explore performance scenarios without the need of implementation.

7. Acknowledgements

Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the US Department of Energy.

8. References

[1] K. Davis, K. Berkbigler, B. Bush et.al. "An Approach to Extreme-Scale Simulation of Novel Architectures", SC2001, Denver, November 2001.

[2] S. Girona, J. Labarta, R.M. Badia, "Validation of Dimemas communication model for MPI collective operations", in Proc. EuroPVM/MPI'2000, Balatonfured, Hungary, September 2000.

[3] J.S. Harper, D.J. Kerbyson, G.R. Nudd, "Analytical Modeling of Set-Associative Cache Behavior", *IEEE Transactions on Computers*, 48(10), October 1999, pp. 1009-1024.

[4] A. Hoisie, O. Lubeck, H. Wasserman, "Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures using Multidimensional Wavefront Applications", *Int. J. of High Performance Computing Applications*, Vol. 14, No. 4, Winter 2000, pp. 330-346.

[5] G. Karypis, V. Kumar, "METIS 4.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System", Technical Report, Department of Computer Science, University of Minnesota, 1998.

[6] D.J. Kerbyson, H.J. Alme, A. Hoisie et. al, "Predictive Performance and Scalability Modeling of a Large-Scale Application", in Proc. SC2001, Denver, November 2001.

[7] D.J. Kerbyson, S.D. Pautz, A. Hoisie, "Predictive Modeling of Parallel Sn Sweeps on Unstructured Meshes", Los Alamos National Laboratory report LA-UR-02-2662, May 2002.

[8] G.R. Nudd, D.J. Kerbyson et.al. "PACE:A Toolset for the Performance Prediction of Parallel and Distributed Systems", *Int. J. of of High Performance Computing Applications,* Vol. 14, No. 3, Fall 2000, pp. 228-251.

[9] S.D. Pautz, "An Algorithm for Parallel Sn Sweeps on Unstructures Meshes", *J. Nuclear Science and Engineering*, Vol. 140, 2002, pp. 111-136.

[10] E. Papaefstathiou, "Design of Performance Technology Infrastructure to Support the Construction of Responsive Software", in: Proc of 2nd ACM Int. Workshop on Software and Performance, Ottawa, Canada, September 2000, pp. 96-104.

[11] F. Petrini, W.C. Feng, A. Hoisie, S. Coll, E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology", *IEEE Micro*, 22(1), 2002, pp. 46-57.